fq

jq for binary formats

Mattias Wadman

Background

- Use various tools to extract data
  - ffprobe, gm identify, mp4dump, mediainfo, wireshark, one off programs, ...
- Convert to usable format and do queries
  - jq, grep, sqlite, sort, awk, sed, one off programs, ...
- Digging into and slicing binaries
  - Hexfiend, hexdump, dd, cat, one off programs, ...
- Personal interest
  - Learn about media, encoding, decoding and binary formats
  - Programming languages

Wishlist

"Want to see everything about this picture except the picture"

- An extremely verbose version of file(1)
- Debugger for files
- Select and query using a language
- Make parts of a file symbolically addressable
- Nested formats and binaries
- Convenient bit-oriented decoder DSL

jq + bit-stream decoder DSL = fq

- Know enough jq to know it would probably fit
- Had experimented with decoder DSL:s
- Possible to combine?
- Did some prototypes and it seems so

jq

"The JSON indenter"

- A tool and a language
- JSON input ➡ jq filter ➡ JSON output
- Syntax is a superset of JSON
  - Any JSON is a valid jq filter
- Functional language based on generators and backtracking
  - Expressions can "output" zero, one or more values
- Implicit input and output similar to shell pipes
- Extraordinary iteration and combinatorial abilities
- Icon and Haskell closest langauge relatives

Examples

```
# Literals
> 123
123

> "abc"
"abc"

> [1,2,3]
[
  1,
  2,
  3
]

> {a: (1+2+3), b: ["abc", false, null]}
{
  "a": 6,
  "b": [
    "abc",
    false,
    null
  ]
}
```

Examples

```
# Pipeline using pipe operator "|" and identity function "." for current input
> "hello" | length | . * 2
10

# Multiple outputs using output operator ","
> 1, 2 | . * 2
2
4

# Index array or object using .[key/index] or just .key for objects
> [1,2,3][1]
2

# Collect outputs into array using [...]
> [1,empty,2]
[1,2]

# Iterate array or object using .[]
> [[1,2,3][]]
[1,2,3]
```

Examples

```
# Generators and backtracking
> 1, (2, 3 | . * 2), 4
1
4
6
4

# Conditional, boolean operators and comparsion
> if 1 == 2 and true then "a" else "b" end
"b"

# Reduce and foreach
> reduce (1,2,3) as $i (0; . + $i)
6
> foreach (1,2,3) as $i (0; . + $i; .)
1
3
6

# Bindings ("variables")
> 1 as $a | 2 as $b | $a + $b
3
```

Examples

```
# Function using lambda argument
# map from standard library:
def map(f): [.[] | f];
> [1,2,3] | map(. * 2)
[
  2,
  4,
  6
]
# select from standard library:
def select(f): if f then . else empty end;
> [1,2,3,4] | map(select(. % 2 == 0))
[
  2,
  4
]

# Function using argument binding and recursion to output multiple values
def down($n):
  if $n >= 0 then $n, down($n-1)
  else empty
  end;
```

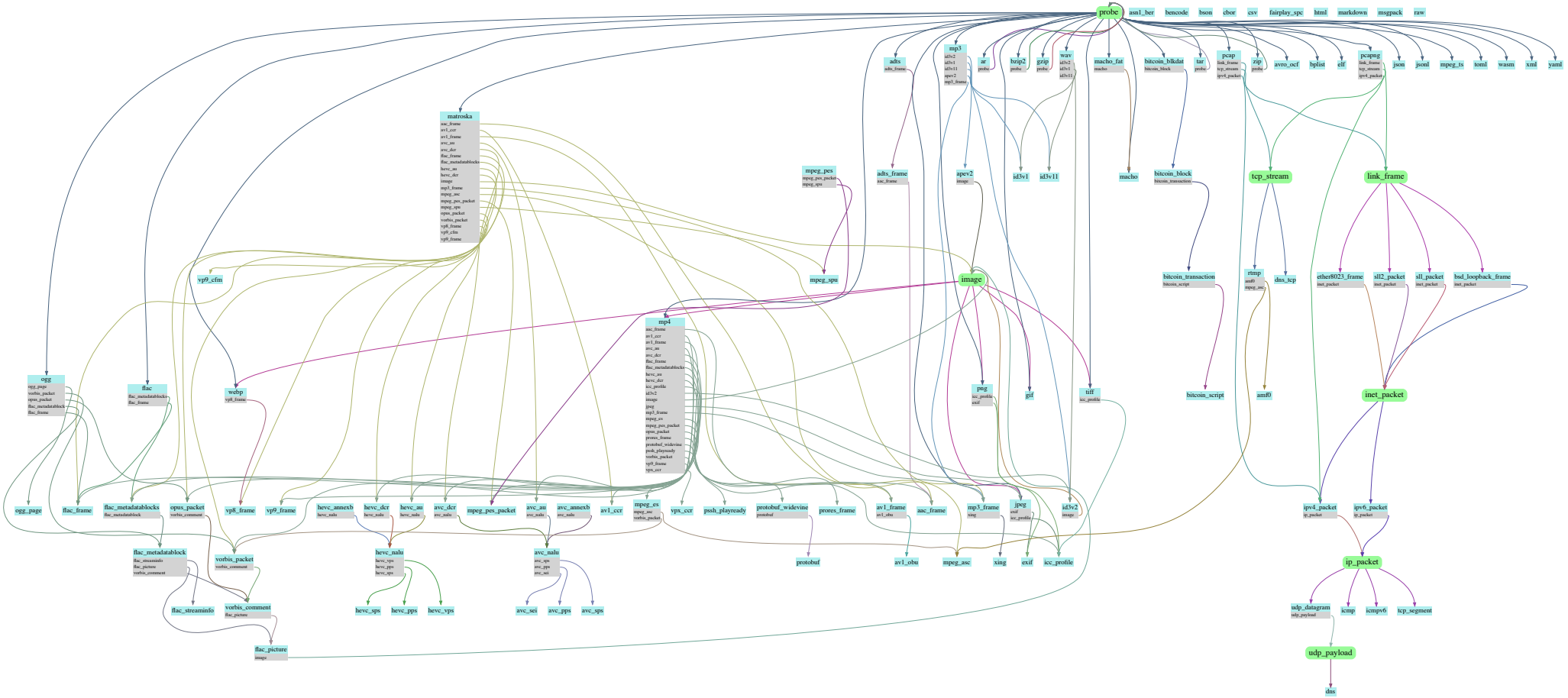
Examples

```
# recurse and ".."
> {a: [1]} | ..
{
  "a": [
    1
  ]
}
[
  1
]
1

def grep_by(f): .. | select(f)?;
> {a: [1,2]} | grep_by(type == "number")
1
2

def noargs: 123;
> noargs
123

def twoargs(a; b): a | b;
> 0 | twoargs(. + 1; . + 2)
3
```

fq

"The binary indenter"

- Superset of jq
- Binary, JSON, XML, ... ➡ jq filter ➡ Binary, fancy hexdump, JSON, XML, ...
- 108 input formats (~50% media related, 32 supports probe)
- Additional standard library functions
- Additional types that act as standard jq types but has special abilities
  - *Decode value* has bit range, actual and symbolic value, description, ...
  - *Binary* has a unit size, bit or bytes, and can be sliced
- Re-implements most of jq's CLI interface
- Interactive REPL with completion and sub-REPL support



aac_frame, adts, adts_frame, amf0, apev2, av1_ccr, av1_frame, av1_obu, avc_annexb, avc_au, avc_dcr, avc_nalu, avc_sei, avro_ocf, exif, fairplay_spc, flac, flac_frame, flac_metadatablock, flac_metadatablocks, flac_picture, flac_streaminfo, gif, hevc_annexb, hevc_au, hevc_dcr, hevc_nalu, icc_profile, id3v1, id3v11, id3v2, jpeg, matroska, mp3, mp3_frame, mpeg_asc, mpeg_spu, mpeg_ts, ogg, ogg_page, opus_packet, png, prores_frame, pssh_playready, rtmp, tiff, vorbis_comment, vorbis_packet, vp8_frame, vp9_cfm, vp9_frame, vpx_ccr, wav, webp, xing

Usage

- Basic usage
  - `fq . file`, `cat file | fq`
- Multiple input files
  - `fq 'grep_by(format == "exif")' *.png *.jpeg`
- Hexdump, JSON and binary output
  - `fq '.frames[10] | d' file.mp3`
  - `fq '[grep_by(format == "dns").questions[].name.value]' file.pcap`
  - `fq 'first(grep_by(format == "jpeg")) | tobytes' file > file.jpeg`
- Interactive REPL
  - `fq -i . *.png`

Some use cases

- Query, aggregate and compare
  - Lookup container metadata
  - How many unique decoder configurations does this mp3 file use
  - Show edit list for a mp4 file
  - List encoder software used for a group of FLAC files
  - Show what is different between two files
- Inspect broken or unknown file
  - Look for truncation or "holes"
  - Try to decode parts
- Assist when developing software (for example fq itself!)
- Basic modification and transmuxing

```
# display a decode value
$ fq . file.mp3
      │00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f│0123456789abcdef│.{}: file.mp3 (mp3)
0x000 │49 44 33 04 00 00 00 00 15 39 54 53 53 45 00 00│ID3.....9TSSE..│  headers[0:1]:
*     │until 0xac2.7 (2755)                           │               │
0xac0 │         ff fb 40 c0 00 00 00 00 00 00 00 00 00│   ..@.........│  frames[0:3]:
0xad0 │00 00 00 00 00 00 00 00 49 6e 66 6f 00 00 00 0f│........Info....│
*     │until 0xd19.7 (end) (599)                      │               │
      │                                               │               │  footers[0:0]:


# expression returning a number
$ fq '.frames | length' file.mp3
3


# raw bytes
$ fq 'grep_by(format == "png") | tobytes' file.mp3 >file.png
$ file file.png
file.png: PNG image data, 320 x 240, 8-bit/color RGB, non-interlaced


# interactve REPL
$ fq -i . file.mp3
mp3> .frames | length
3
mp3> .header[0] | repl
> .headers[0] id3v2> .frames[0].text
      │00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f│0123456789abcdef│
0x10  │                4c 61 76 66 35 38 2e 37 36 2e 31│     Lavf58.76.1│.headers[0].frames[0].text: "Lavf58.76.100"
0x20  │30 30 00                                       │00.             │
> .headers[0] id3v2> .frames[0].text | tovalue
"Lavf58.76.100"
> .headers[0] id3v2> ^D
mp3> ^D
$
```

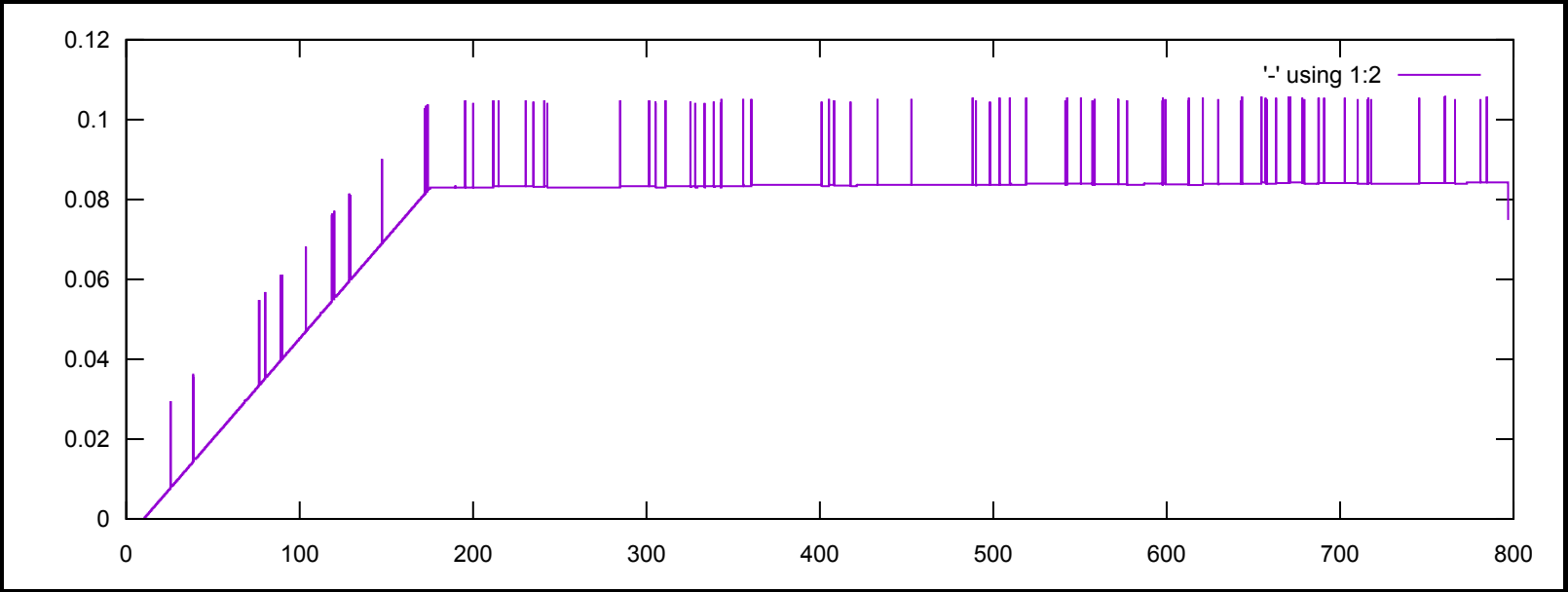
```
$ fq . test.mp4
       |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f|0123456789abcdef|.{}: test.mp4 (mp4)
0x0000 |00 00 00 20 66 74 79 70 69 73 6f 6d 00 00 02 00|... ftypisom....|  boxes[0:4]:
*      |until 0x4975.7 (end) (18806)
0x0030 |00 00 02 ad 06 05 ff ff a9 dc 45 e9 bd e6 d9 48|..........E....H|  tracks[0:2]:
*      |until 0x4975.7 (end) (18758)

$ fq d test.mp4
       |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f|0123456789abcdef|.{}: test.mp4 (mp4)
       |                                               |                |  boxes[0:4]:
       |                                               |                |    [0]{}: box
0x00000|00 00 00 20                                    |...             |      size: 32
0x00000|            66 74 79 70                       |    ftyp        |      type: "ftyp" (File type and compatibility
0x00000|                        69 73 6f 6d            |        isom    |      major_brand: "isom"
0x00000|                                    00 00 02 00|            .... |      minor_version: 512
       |                                               |                |      brands[0:4]:
0x00010|69 73 6f 6d                                    |isom            |        [0]: "isom" (All files based on the ISO
0x00010|            69 73 6f 32                       |    iso2        |        [1]: "iso2" (All files based on the 200
0x00010|                        61 76 63 31            |        avc1    |        [2]: "avc1" (Advanced Video Coding exte
0x00010|                                    6d 70 34 31|            mp41 |        [3]: "mp41" (MP4 version 1)
       |                                               |                |    [1]{}: box
0x00020|00 00 00 08                                    |....            |      size: 8
0x00020|            66 72 65 65                       |    free        |      type: "free" (Free space)
       |                                               |                |      data: raw bits
       |                                               |                |    [2]{}: box
0x00020|                        00 00 40 b2            |        ..@.    |      size: 16562
0x00020|                                    6d 64 61 74|            mdat |      type: "mdat" (Media data container)
0x00030|00 00 02 ad 06 05 ff ff a9 dc 45 e9 bd e6 d9 48|.........E....H |      data: raw bits
*      |until 0x40d9.7 (16554)
       |                                               |                |    [3]{}: box
0x040d0|                                00 00 08 9c    |        ....    |      size: 2204
0x040d0|                                            6d 6f|            mo |      type: "moov" (Container for all the meta-
0x040e0|6f 76                                          |ov              |
       |                                               |                |      boxes[0:4]:
       |                                               |                |        [0]{}: box
0x040e0|      00 00 00 6c                              |  ...l          |          size: 108
0x040e0|                  6d 76 68 64                  |     mvhd       |          type: "mvhd" (Movie header, overall d
…
```

```
# use ffprobe and jq to figure out aspect ratio
$ ffprobe -v quiet -i test.mp4 -show_streams -of json | jq '.streams[] | select(.codec_type == "video") | .width / .heig
1.3333333333333333

$ fq 'grep_by(.type=="trak") | select(grep_by(.type=="hdlr").component_subtype == "vide") | grep_by(.type=="tkhd") | .tr
1.3333333333333333

$ cat mp4.jq
def mp4_trak($subtype):
  ( grep_by(.type=="trak")
  | select(grep_by(.type=="hdlr").component_subtype == $subtype)
  );

$ fq -L . 'include "mp4"; mp4_trak("vide") | grep_by(.type=="tkhd")' test.mp4
       |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f|0123456789abcdef|.boxes[3].boxes[1].boxes[0]{}: box
0x41a0|                              00 00 00 5c       |           ...\  |  size: 92
0x41a0|                                       74 6b    |              tk |  type: "tkhd" (Track header, overall informatio
0x41b0|68 64                                           |hd               |
0x41b0|      00                                        |  .              |  version: 0
0x41b0|         00 00 03                              |    ...          |  flags: 3
0x41b0|                  00 00 00 00                   |       ....      |  creation_time: 0 (1904-01-04T00:00:00Z)
0x41b0|                              00 00 00 00       |           ....  |  modification_time: 0 (1904-01-04T00:00:00Z)
0x41b0|                                       00 00    |              .. |  track_id: 1
0x41c0|00 01                                           |..               |
0x41c0|      00 00 00 00                               |  ....           |  reserved1: 0
0x41c0|                  00 00 03 e8                   |       ....      |  duration: 1000
0x41c0|                              00 00 00 00 00 00 |           ......|  reserved2: raw bits
0x41d0|00 00                                           |..               |
0x41d0|      00 00                                     |  ..             |  layer: 0
0x41d0|            00 00                              |    ..           |  alternate_group: 0
0x41d0|                  00 00                         |      ..         |  volume: 0
0x41d0|                        00 00                   |        ..       |  reserved3: 0
0x41d0|                              00 01 00 00 00 00 |          ......|  matrix_structure{}:
0x41e0|00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00|............    |
0x41f0|00 00 00 00 00 00 00 00 00 00 40 00 00 00       |..........@...   |
0x41f0|                                       01 40    |              .@ |  track_width: 320
0x4200|00 00                                           |..               |
0x4200|      00 f0 00 00                               |  ....           |  track_height: 240
```

```
$ fq -L . 'include "mp4"; mp4_trak("vide") | grep_by(.type=="tkhd") | .track_width / .track_height' test.mp4
1.3333333333333333

$ fq -L . 'include "mp4"; mp4_trak("vide") | grep_by(.type=="tkhd") | {track_width, track_height}' test.mp4
{
  "track_height": 240,
  "track_width": 320
}

$ fq -L . 'include "mp4"; mp4_trak("vide") | grep_by(.type=="tkhd") | {width: .track_width}' test.mp4
{
  "width": 320
}
```

```
$ fq -L . 'include "mp4"; mp4_trak("soun","vide") | grep_by(.type=="mdhd")' test.mp4
       |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f|0123456789abcdef|.boxes[3].boxes[2].boxes[2].boxes[0]{}: box
0x4610|          00 00 00 20                           |   ...          |  size: 32
0x4610|                   6d 64 68 64                  |       mdhd     |  type: "mdhd" (Media header, overall informatio
0x4610|                               00               |            .   |  version: 0
0x4610|                                  00 00 00      |               ...  flags: 0
0x4610|                                           00   |                  .  creation_time: 0 (1904-01-04T00:00:00Z)
0x4620|00 00 00                                        |...             |
0x4620|          00 00 00 00                           |   ....         |  modification_time: 0 (1904-01-04T00:00:00Z)
0x4620|                   00 00 ac 44                  |       ...D     |  time_scale: 44100
0x4620|                               00 00 b0 44      |           ...D |  duration: 45124
0x4620|                                        55      |              U |  language: "und"
0x4630|c4                                              |.               |
0x4630|   00 00                                        | ..             |  quality: 0
       |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f|0123456789abcdef|.boxes[3].boxes[1].boxes[2].boxes[0]{}: box
0x4230|       00 00 00 20                              |   ...          |  size: 32
0x4230|                0x64 68 64                      |       mdhd     |  type: "mdhd" (Media header, overall informatio
0x4230|                            00                  |            .   |  version: 0
0x4230|                               00 00 00         |               ...  flags: 0
0x4230|                                        00 00   |                  ..  creation_time: 0 (1904-01-04T00:00:00Z)
0x4240|00 00                                           |..              |
0x4240|       00 00 00 00                              |   ....         |  modification_time: 0 (1904-01-04T00:00:00Z)
0x4240|                0x00 32 00                      |       ..2.     |  time_scale: 12800
0x4240|                            00 00 32 00         |           ..2. |  duration: 12800
0x4240|                                        55 c4   |              U.|  language: "und"
0x4250|00 00                                           |..              |  quality: 0

$ fq -L . 'include "mp4"; mp4_trak("soun","vide") | grep_by(.type=="mdhd") | .duration / .time_scale' test.mp4
1.023219954648526
1
$ fq -n -L . 'include "mp4"; [inputs | {(input_filename): (mp4_trak("vide") | grep_by(.type=="mdhd") | .duration / .time
{
  "big_buck_bunny.mp4": 60.095,
  "test.mp4": 1
}
```

```
#!/usr/bin/env fq -r -d mp4 -o decode_samples=false -f
# plot aac drift, assumes 1024 samples per packet and 44100Hz sample rate
# $ ./drift.jq drift.mp4 | gnuplot"
# ...
# 13.003174603174603 0.0013378684807256237
# 13.02639455782313 0.0013378684807256237

( nth(1; grep_by(.type == "stts"))
| [ .entries[]
  | range(.count) as $_
  | .delta
  ]
| [foreach .[] as $n (0; .+($n-1024);.)] as $d
| range(length)
| "\((.*1024)/44100) \($d[.]/44100)"
)
```



Binary and binary array

- A binary is created using `tobits`, `tobytes`, `tobitsrange` or `tobytesrange`.
  - From decode value `.frames[1] | tobytes`
  - String or number `"hello" | tobits`
  - Binary array `[0xab, ["hello", .name]] | tobytes`
- Can be sliced using normal jq slice syntax.
  - `"hello" | tobits[8:8+16]` are the bits for `"el"`
- Can be decoded
  - `[tobytes[-10:], 0, 0, 0, 0] | flac_frame`

Example queries

- Slice and decode
  - `tobits[8:8+8000] | mp3_frame | d`
  - `match([0xff,0xd8]) as $m | tobytes[$m.offset:] | jpeg`
- ASN1 BER, CBOR, msgpack, BSON, ... has `torepr` support
  - `fq -d cbor torepr file.cbor`
  - `fq -d msgpack '[torepr.items[].name]' file.msgpack`
- PCAP with TCP reassembly, look for GET requests
  - `fq 'grep("GET .*")' file.pcap`
- Parent of scalar value that includes bit 100
  - `grep_by(scalars and in_bits_range(100)) | parent`

Use as script interpreter

```
#!/usr/bin/env fq -d mp4 -f

( first(.boxes[] | select(.type == "moov")?)
| first(.boxes[] | select(.type == "mvhd")?) as $mvhd
| { time_scale: $mvhd.time_scale,
    duration: ($mvhd.duration / $mvhd.time_scale),
    tracks:
      [ .boxes[]
      | select(.type == "trak")
      | [("mdhd", "stsd", "elst") as $t | first(grep_by(.type == $t))] as [$mdhd, $stsd, $elst]
      | { data_format: $stsd.boxes[0].type,
          media_scale: $mdhd.time_scale,
          edit_list:
            [ $elst.entries[]
            | { track_duration: (.segment_duration / $mvhd.time_scale),
                media_time: (.media_time / $mdhd.time_scale)
              }
            ]
        }
      ]
  }
)
```

Use as script interpreter

```
$ ./editlist file.mp4
{
  "duration": 60.095,
  "time_scale": 600,
  "tracks": [
    {
      "data_format": "mp4a",
      "edit_list": [
        {
          "media_time": 0,
          "track_duration": 60.095
        }
      ],
      "media_scale": 22050
    },
    {
      "data_format": "avc1",
      "edit_list": [
        {
          "media_time": 0,
          "track_duration": 60.095
        }
...
```

Decode API

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

Decode API

SPS HRD parameters from ITU-T H.264 specification

```
func avcHdrParameters(d *decode.D) {
    cpbCnt := d.FieldUFn("cpb_cnt", uEV, scalar.UAdd(1))
    d.FieldU4("bit_rate_scale")
    d.FieldU4("cpb_size_scale")
    d.FieldArray("sched_sels", func(d *decode.D) {
        for i := uint64(0); i < cpbCnt; i++ {
            d.FieldStruct("sched_sel", func(d *decode.D) {
                d.FieldUFn("bit_rate_value", uEV, scalar.UAdd(1))
                d.FieldUFn("cpb_size_value", uEV, scalar.UAdd(1))
                d.FieldBool("cbr_flag")
            })
        }
    })
    d.FieldU5("initial_cpb_removal_delay_length", scalar.UAdd(1))
    d.FieldU5("cpb_removal_delay_length", scalar.UAdd(1))
    d.FieldU5("dpb_output_delay_length", scalar.UAdd(1))
    d.FieldU5("time_offset_length")
}
```

Decode API

Formats can use other formats. Simplified version of mp3 decoder:

```
func decode(d *decode.D, in interface{}) interface{} {
    d.FieldArray("headers", func(d *decode.D) {
        for !d.End() {
            d.TryFieldFormat("header", headerGroup)
        }
    })

    d.FieldArray("frames", func(d *decode.D) {
        for !d.End() {
            d.TryFieldFormat("frame", mp3Group)
        }
    })

    d.FieldArray("footers", func(d *decode.D) {
        for !d.End() {
            d.TryFieldFormat("footer", footerGroup)
        }
    })

    return nil
}
```

Future

- Declarative decoding like kaitai struct, decoder in jq
- Nicer way to handle checksums, encoding, validation etc
- Schemas for ASN1, protobuf, ...
- Better support for modifying data
- More formats like tls, http, http2, grpc, filesystems, ...
- Encoders
- More efficient, lazy decoding, smarter representation
- GUI
- Streaming input, read network traffic `tap("eth0") | select(...)`?
- Hope for more contributors

Thanks and useful tools

- @itchyny for gojq
- Stephen Dolan and others for jq
- HexFiend
- GNU poke
- Kaitai struct
- Wireshark
- vscode-jq (https://github.com/wader/vscode-jq)
- jq-lsp (https://github.com/wader/jq-lsp)

Thank you

jq for binary formats

Mattias Wadman
mattias.wadman@gmail.com (mailto:mattias.wadman@gmail.com)
https://github.com/wader/fq (https://github.com/wader/fq)
@mwader (http://twitter.com/mwader)